On-Line Recognition of Mathematical Expressions Using Automatic Rewriting Method

T. Kanahori[1], K. Tabata[1], W. Cong[2], F.Tamari[2], and M. Suzuki[1]

[1] Graduate School of Mathematics, Kyushu University 36,
Fukuoka, 812–8581 Japan
suzuki@math.kyushu-u.ac.jp
[2] Fukuoka University of Education
729 Akama, Munakata-shi, Fukuoka, 811–41 Japan,
tamari@fukuoka-edu.ac.jp

Abstract. This paper describes our system of on-line recognition of mathematical expressions. Users can input mathematical expressions by handwriting. As soon as a character is written, it is rewritten by neat strokes in an appropriate position and size automatically. This *Automatic Rewriting Method* improves the accuracy of the structure analysis of the written mathematical expressions. The written mathematical expressions can be output into files in the notation of LaTeX and MathML. By this handwriting interface, the system realizes a very easy intuitive methods to input mathematical expressions into computer.

1 Introduction

Recently, the use of computer and network is becoming widely spread. However, it is true that the user interfaces of current computer systems are not convenient to input mathematical expressions (see [1], [2] and [3]). For example, the widely used data format TeX requires some learning to master the notations, and it is not easy to understand the meaning of the written expressions by the TeX source at a glance. To realize easier treatment of mathematical expressions of various formats, we are developing a new handwriting interface to input mathematical expressions into computer. Users can input mathematical expressions by handwriting in the *handwriting area*, and edit the input expressions on the *display area*. The edited mathematical expressions can be output into a file in the notation of LaTeX or MathML.

Presently, this system can recognize all the alphanumeric characters, almost all Greek letters, and other symbols frequently used in mathematical expressions, and can analyze the expressions used in high school mathematics or in the first course of university mathematics, including fractions, square roots, subscripts, superscripts, integrals, limits, summations and the function names ‘lim’, ‘log’, ‘cos’, ‘sin’, ‘tan’, etc. The matrices are excluded at the moment. The structure of mathematical expressions may have the nested structures. However, deeply nested structure leads to the increase of small size characters and naturally increases the errors of the recognition.

In this paper, we describe our method of the recognition of handwritten characters and the algorithms of *Automatic Rewriting Method*, and report the performance of our experimental system.

2 Outline of The Experimental System

The main window of the system consists of two areas (see Figure 1). The upper area is the display area and the lower is the handwriting area. In the handwriting area, users write mathematical expressions by using a mouse, a data tablet or a pen display.

As soon as a character is written, it is recognized and rewritten by neat strokes in an appropriate size and position automatically. Pushing the button ⟨OK⟩, the expression is analyzed, and the result is displayed in type setting form in the display area.

In the display area, we can edit mathematical expressions using ordinary editing operations: selection, cut, copy, paste and delete. The users can save them into a file in the notations of LaTeX or MathML.

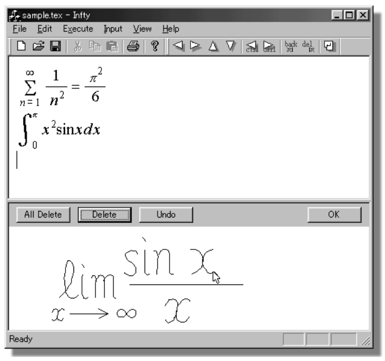
sample.tex - Infty
File Edit Execute Input View Help
$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$
$\int_0^{\pi} x^2 \sin x dx$
All Delete
Delete
Undo
OK
$\lim_{x \to \infty} \frac{\sin x}{x}$
Ready

Fig. 1. The experimental system

3 Character Recognition

In this section, we describe our method of character recognition in our handwriting interface. We implemented two different methods of character recognition, to get recognition results by *voting*. One of the recognition methods uses the distribution of the 8-direction elements of the strokes on 3×5 meshes of the he character rectangle (Section 3.1). The other method uses the matching of segmented stroke sequence (Section 3.2). Each of the two recognition methods returns three ordered candidates with costs. The *voting cost* of the candidate is taken to be the ratio of its cost to the third candidate's, and the final recognition results is determined by the ascending order of the sum of their two voting costs of the two recognition methods.

The characters and symbols recognized in this system include all the alphanumeric characters, some Greek letters, and other symbols frequently used in mathematical expressions (see [3]).

3.1 Direction Element Feature

To extract this feature, we first take a normalized coordinate system which squarely converts the bounding rectangle of a character, and subdivide the bounding rectangle into 3×5 meshes (3 meshes in the horizontal direction, and 5 meshes in the vertical direction).

Let $d_0, d_1, d_2, \ldots, d_7$ be the directions taken from the x-axial direction to the backing every 45 degree (see Figure 2). Given a segment of length L of direction between d_i, d_{i+1} making the angle θ_1, θ_2 respectively with them, the contribution of the segment to the directions d_i, d_{i+1} are defined by

$$L_i = \frac{L\theta_2}{\theta_1 + \theta_2}, \quad L_{i+1} = \frac{L\theta_1}{\theta_1 + \theta_2},$$

respectively, where $L_8 = L_0$ and $d_8 = d_0$. L_i is called the *direction component* of the segment to d.

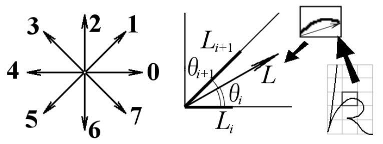

Fig. 2. The direction component

Calculating the direction components of each segment which constitutes the strokes of a character, these components are distributed into the 3×5 meshes defined above. Thus we obtain a feature vector of dimension $3 \times 5 \times 8 = 120$, which we call the *direction element* feature([4]). Since we take a coarse mesh (3×5), this feature vector has a robust property for the distortion of the character shape.

3.2 Matching of Segmented Stroke Sequence

If a written stoke has some small loops, they are modified to cusps (see Figure 3). After this modification, the written stroke is segmented at extreme points on the vertical coordinate. The stroke is segmented at cusp points again (see Figure 3). Segment strokes which can be regarded as a straight line are classified into 8 patterns (8 directions in Figure 2), and segment strokes which are winding down strokes are classified into 10 patterns (see Figure 4). Segment strokes which are winding upstrokes are also classified similarly. Hence, there are $8 + 10 \times 2 = 28$ patterns of segment strokes. The character recognition is done by the matching of the sequences of segment stroke patterns thus obtained.

To calculate the recognition cost, the following features are used: segment strokes' aspect ratios, the positions of their bounding rectangles, the directions of their original vectors v_o and terminal vectors v_t (see Figure 5). The cost of a written stroke for a candidate is determined by the sum of the differences of these features between the corresponding segment strokes.

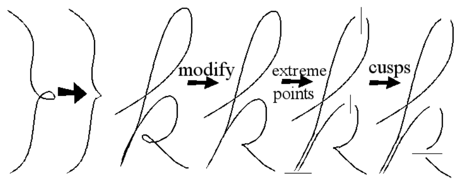

Fig. 3. Modification and Segmentation

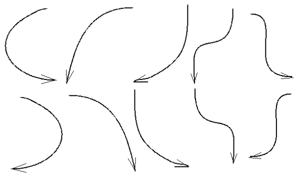

Fig. 4. Downstrokes

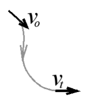

Fig. 5. v_o, v_t

4 Automatic Rewriting Method

The distortion of input characters and the turbulence of the positions or the scales of characters usually cause serious difficulties in the structure analysis of mathematical expressions, in which the positions and the scales of the characters have special meanings. An error of a character recognition or of the segmentation of the strokes into character units leads sometimes to a fatal error of the structure analysis of the mathematical expression. The labors for the correction of this kind of errors disturb seriously the smooth input of mathematical expressions.

Our automatic rewriting method is introduced in order to overcome this difficulty. In this method, whenever the pen is up, the strokes are recognized. Each recognized character is rewritten by neat strokes in an appropriate size and position automatically (see Figure 1). By this rewriting, the user can identify each recognition error immediately when it occurs, and can correct it easily.

In this section, we shall explain the algorithms to determine characters and to select appropriate positions and sizes of the determinate characters.

4.1 Determination of Characters

When a stroke is written, it is necessary to decide whether a character is fully written up or not at each time the pen is up in our method. To explain the Determination Algorithm for this operation, we classify the results of the character recognition into two groups. One is the group of characters, named *extendable characters*, which can be extended to other character by adding some strokes. For example, 'F' is extendable to 'E', 'C' to 'G' or 'd', and '=' to '≠'.

The other is the group of characters, named *unextendable characters*, which can not be extended to any other character by adding strokes. For example, 'E' and 'G' are unextendable.

Each extendable character has extendable areas, where a next stroke is expected to be pushed down (see Figure 6). The determination algorithm of character unit proceeds as follows:

1. Let S be empty. (S means the sequence of untreated strokes.)
2. When a pen is up, add the written stroke to S
3. Let R be the recognition result considering S as one character.
4. If R is unextendable, then output R as the determined character of the strokes S and go to the step 1.
5. Wait for the next stroke N to be written. If N started from R's extendable area within 2 seconds, then go to step 2, else output R as the determined character of the strokes S and go to step 1.

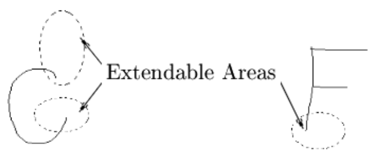

Fig. 6. Extendable areas

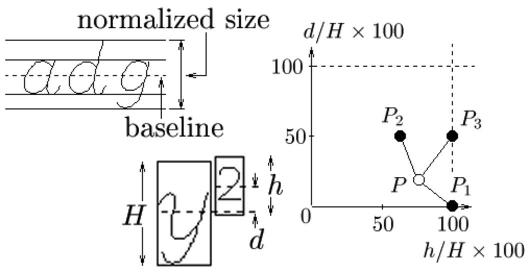

Fig. 7. Decision of Position and Sizes

4.2 Determination of Positions and Sizes of Characters

A structure of mathematical expression can be represented by a tree structure. Corresponding to it, the notion of *parent (or child) character* is introduced. For the expression $x^a + y^b$', for example, 'x' and y are the parent characters of 'a' and b with the relation 'superscript', and x is also the parent of '+' with the relation 'same line'. The position and the size of an input character are determined together with its parent character.

Let C be an input character, and D be the candidate of the parent character of C. Then, the *relation point* $P(C, D)$ is defined by

$$P(C, D) := (h/H \times 100, d/H \times 100),$$

where h is the normalized size of C, H is the normalized size of D and d is the distance of the baselines of C and D (see Figure 7). The *cost* between C and D on **Relation** i, $c_i(C, D)$, is defined by

$$c_i(C, D) := d(P_i, P(C, D)) \quad (i = 1, 2, 3),$$

where **Relation 1** means that C and D are on the same line, **Relation 2** means that C is either the superscript or the subscript of D, **Relation 3** means that C is either the numerator or the denominator of the fraction on the same line as D's, and P_i are ideal relation points on **Relation** i. In this system, $P_1 = (100, 0)$, $P_2 = (60, 50)$ and $P_3 = (100, 50)$.

The algorithm to find the parent character M of the new input character C and to obtain their relation R is as follows:

1. Let M and R be NULL. If C is the first input character in the mathematical expression, then quit.
2. Let D be the input character just before C, min be a positive number which is large enough, and set n=0.
3. If D is NULL then quit.
4. If the cost $c_1(C, D)$ is small enough, then let $M = D$ and $R =$ **Relation 1** and quit.
5. Let $m = \text{argmin}\{k \neq n | c_k(C, D)\}$. If $c_m(C, D)$ is smaller than min, then let $min = c_m(C, D)$, $M = D$, and $R =$ **Relation** m. Let further **Relation** n be the relation between D and its parent character, and replace D by the parent character of D. Go to the step 3.

The size and the position of the input character C is determined by the pair of its parent character M the relation R thus obtained.

4.3 Structure Analysis of Mathematical Expressions

By our Automatic Rewriting Method, the recognition result of each input character and its position and size are corrected by the user, as soon as an error occurs. Consequently, the structure analysis of the mathematical expression proceeds with very high accuracy. In practice, no error occurs in the structure analysis expect for the case where the input rule is ignored intentionally.

5 Experimental Results

To evaluate the efficiency of Automatic Rewriting Method and the performance of our recognition engine, the experiments were carried out using thirty writers who had got their hands in writing mathematical expressions, but never used our system in the following order:

Experiment 1 First, we explained how to use our system, following the manual prepared for this experiment (for about 10 minutes for each writer). Then, each writer wrote the following mathematical expressions (1)$\sim$(7),

$$(1)\quad ax^2+bx+c=0,\quad (2)\quad \frac{s+t}{p+q},\quad (3)\quad e^{-\frac{x^2}{2}},\quad (4)\quad x=\frac{-b\pm\sqrt{b^2-4ac}}{2a},$$

$$(5)\quad \sum_{i=1}^{n} i=\frac{n(n+1)}{2},\quad (6)\quad \lim_{x\to 0}\frac{\sin x}{x},\quad (7)\quad \int_a^b \log x dx,$$

and we counted the number of times of error correction actions of the writer. For the correction of character recognition error, the writers are instructed to switch the recognition result with its next candidate by clicking on the character, and to delete last one character by pushing the button ⟨Delete⟩ only if there is no correct result among the three candidates. On the other hand, for the case the size or the position of the rewritten character is not appropriate, or the character segmentation is wrong, the writers are instructed to push first the button ⟨Delete⟩. In this way, the error correction actions are classified by their causes into four groups: **1. order**: the first candidate of the character recognition was not correct, **2. candidate**: no correct result was found among the three candidates, **3. position**: the position or the size of the rewritten character was wrong, **4. segmentation**: one character was recognized as several characters. For each written character, we logged its recognition cost, the distance between baselines of the character and its rewritten character. We did this process 6 times. Finally, we reshuffled the above expressions and did the similar experiment as 7th process.

Experiment 2 After Experiment 1, each writer wrote the following expressions (i)$\sim$(iv), which are more complicated than Experiment 1's. Then, we counted and logged in the similar way to Experiment 1. We did this process 3 times.

$$\text{(i)}\ \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n=e,\qquad \text{(ii)}\ \int_0^\infty \frac{\log x}{1+x^2}dx,\qquad \text{(iii)}\ \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)^2},$$

$$\text{(iv)}\ \int_0^\infty e^{-a^2x^2}dx=\frac{\sqrt{\pi}}{2|a|}$$

Experiment 3 After Experiment 2, we told each writer to write as fast as possible each the expressions (i)$\sim$(iv). For each expression, we counted the time from the first stroke's beginning until the whole expression exactly written up, including the time to correct.

Experiments 1$\sim$3 are intended to evaluate the effects of Automatic Rewriting Method. After these experiments, we did the following experiment to evaluate the recognition rate of our recognition engine.

Experiment 4 For the training, each writer wrote all alphabets and numerics 3 times. Then, each writer wrote 20 characters (capital:small:numeric = 8:8:4) which we had chosen randomly, and the number of recognition errors is counted. We did this process 5 times, and classified the number of errors into two cases: 1. the correct result was not the first candidate, but in the first three candidates, 2. the correct result was not in the first three candidates. Moreover, we also classified lower-case letters into two classes according as the writers had written in Experiment 1 or 2, or not, and counted the number of errors for each classes.

Figure 8 shows the result of Experiment 1 and 2. In Figure 8, the x-axis means the process times of the experiment, and the y-axis means ratio of correction time to the total number of characters. For example, the ratio of the case **order** is calculated by $(1/N)\sum_{i=1}^{7} e_i$, where $e_i (i = 1, \cdots, 7)$ is the number of errors **order** at the expression (i), and N is the total number of characters in all of the expressions (1)$\sim$(7).

Figure 8 shows that the numbers of correction except **order**'s decrease as the number of writing increases. The **order** correction increases from 4th time. This means that the concentration's slip or the habituation of the writer causes rough writing. But, the decrease of the other corrections shows that this system can cope with the 'rough writing'. Moreover, the **position** correction (or **segmentation**) which leads a fatal error of the structure analysis of mathematical expressions is about 3 times (resp. 4 times) per 100 characters at 7th writing. Hence, the Automatic Rewriting Method is effective to on-line recognition of mathematical expressions.

We expected another advantage of Automatic Rewriting Method as below: The rewritten characters may serve as a model of the feasible character forms and the size for the recognition, and leads the user to a neater writing style. However, there was no major difference in the log of the costs and the distances between baselines from the 1st writing to the 7th in Experimental 1 and 2. Therefore, we could not see the advantage from this point of view.

Figure 9 shows the result of Experiment 3. It shows that, everyone wrote up all the expressions (i)$\sim$(iv) between 1.5 minutes and 4 minutes. For 5 experts of TEX, who had written at least 5 articles of mathematics using TEX, we counted the times to input the same expressions in the notation of TEX. Then, the fastest time was about 1.5 minutes, the latest was about 3.5 minutes, and the average was 1 minute and 58 seconds. This shows that beginners can input complicated mathematical expressions smoothly with simple training, by using our interface, as well as inputting of the experts in the notation of TEX.

Table 1 shows the result of Experiment 4. In Table 1, the 1st row shows the recognition rate of the first candidate and the 2nd row shows the rate in the first three candidates. The 4th column 'appear' (resp. the 5th column 'not appear') means that the rate is for characters which appeared (resp. did not appear) in Experiment 1 or 2. The 1st rate of 'appeared' characters is worst.

This shows the similar result to Graph 8, 'rough writing' by the concentration' slip or habituation.

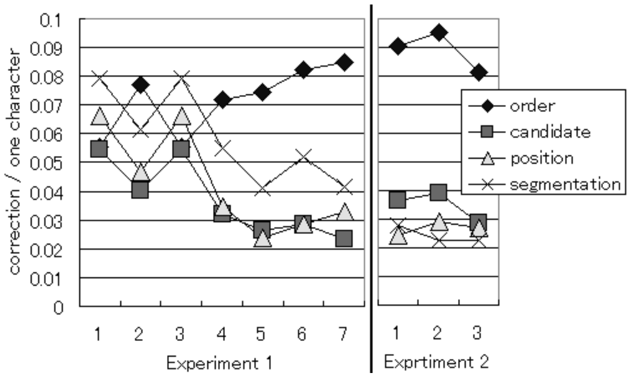

Fig. 8. The ratios of correction

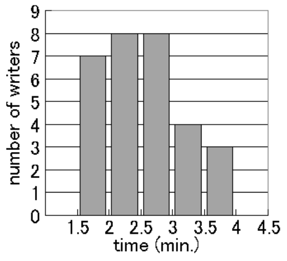

Fig. 9. Writing time

	capital	small	numeric	appear	not appear	total
at the first candidate	95.5%	91.7%	94.5%	90.9%	93.4%	93.8%
in three candidates	97.8%	97.1%	97.3%	97.2%	96.6%	97.4%

Table 1. The rates of current result

6 Conclusion

In this paper, we introduced our system of on-line recognition of mathematical expressions. We proposed Automatic Rewriting Method to improve the accuracy of the structure analysis of handwritten mathematical expressions and to realize an easy and prompt correction method of recognition errors. We emphasized that an easy correction method of the recognition results is extremely important to realize smooth writing of mathematical expressions. From the experimental result, In the experiment, we observed the efficiency of our method to input mathematical expressions by handwriting smoothly with minimum training.

References

1. D. Blostein and A. Grbavec, "Recognition of Mathematical Notation", *Handbook of Character Recognition and Document Image Analysis*, (1997) 557-582.
2. T. Sakurai, Y. Zhao, H. Sugiura and T. Torii, "A Front-end Tool for Mathematical Computation and Education in a Network Environment", *Proc. 3rd. Asian Technology Conference in Mathematics*, Springer (1998) 197-205.
3. H. Okamura, T. Kanahori, W. Cong, R. Fukuda, F. Tamari and M. Suzuki, "Handwriting Interface for Computer Algebra Systems", *Proc. 4th. Asian Technology Conference in Mathematics*, December (1999) 291-300.
4. N. Sun , T. Tabara, H. Aso and M. Kimura, "Printed Character Recognition Using Directional Element Feature ", **IEIEC J74-D-II**, 3, 1991, 330-339 (in Japanese).